Loan Prediction Project

ASU, Faculty of Computer Science

By: *Ali Abdallah*

With supervision of TA. Verena Nashaat

Artificial Intelligence

1.Introduction

Addressing the problem

By the evolution of the banking sector lots of people are applying for bank loans but the bank has its limited assets which it must grant to limited and specific people only, so finding out to whom the loan can be granted which will be a safer option for the bank is a necessary process. So, in this project we try to reduce this risk factor behind selecting the reliable person to save lots of bank efforts and assets. This is done by mining the Big Data of the records of the people to whom the loan was granted before and based on these records/experiences the machine was trained using the machine learning model which give the most accurate result. The main objective of this project is to predict whether assigning the loan to the applicant will be safe or not. This report is divided into four sections:

1. Data Collection.
2. Machine learning models applied on the collected data.

In this report we are going to predict the loan data by using the machine learning algorithms:

- Logistic Regression.
- SVM.
- Decision Tree.
- K-Nearest Neighbors.

2.1 Data Collection

2.1.1 Overview of the data

Before jumping on to displaying the dataset we are working with initially we need various libraries to be imported for designing a code that performs training and testing tasks, Predictions, array operations and so on, they are as follows:

[1]:
```
# Importante Required Libraries
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
```

[2]:
```
import warnings
warnings.filterwarnings("ignore")
```

Now that we have our libraries the 1st step is to look at the data we're working with. Realistically, most of the data we will get, even from the government, can have errors, and it's important to identify these errors before spending time analyzing the data. Normally, we must answer the following questions:

o Do we find something wrong in the data?

o Are there ambiguous variables in the dataset?

o Are there variables that should be fixed or removed?

Let's start by reading the data using the function read.csv() and summarize both datasets:

[3]:
```
# Read The Dataset
data = pd.read_csv('loan_data.csv')
print(data.head())
```

```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
```

[4]:
```
print('Column Names :\n', data.columns)
```

```
Column Names :
 Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

2.1.2 Data cleaning and processing:

In Data cleaning the system detects and correct corrupt or inaccurate records from database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying or detecting the dirty or coarse data. In Data processing the system convert data from a given form to a much more usable and desired form i.e., make it more meaningful and informative.

After the system reads the data, we must label encode the data which means converting the target variable into a numeric form to convert it to a machine-readable form, and this can be done through using LabelEncoder () from Sklearn.preprocessing library.

[5]:
```
#Label Encode The Target Variable
encode = LabelEncoder()
data.Loan_Status = encode.fit_transform(data.Loan_Status)
```

[6]:
```
print(data.head())
```

```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0

   Credit_History Property_Area  Loan_Status
0             1.0         Urban            1
1             1.0         Rural            0
2             1.0         Urban            1
3             1.0         Urban            1
4             1.0         Urban            1
```

The next thing we are going to do is to drop the rows that have NULL values in our dataset.

[7]:
```
print('Shape of data before dropping the rows that have null values :''\n', data.shape)
```

```
Shape of data before dropping the rows that have null values :
 (614, 13)
```

[8]:
```
# Dropping The Rows That Have Null Values
data.dropna(inplace=True)
```

[9]:
```
print('Shape of data after dropping the rows that have null values :''\n', data.shape)
```

```
Shape of data after dropping the rows that have null values :
 (480, 13)
```

2.1.2.1 Determine the training and testing data:

Typically, Here the system separate a dataset into a training set and testing set ,most of the data use for training ,and a smaller portions of data is use for testing. after a system has been processed by using the training set, it makes the prediction against the test set.

[10]:
```
# Train-Test-Split
train, test = train_test_split(data,test_size=0.3,random_state=0)
```

2.1.3 Structured Analysis Planning

Moving on the first thing we need to do and before jumping to analyze the data is to understand the problem statement and create a S.M.A.R.T objective. The next step is to identify our independent variables and our dependent variable.

[23]:
```
# Seperate The dependent variable and Independent variables
train_x = train.drop(columns=['Loan_ID','Loan_Status'])
train_y = train['Loan_Status']
test_x = test.drop(columns=['Loan_ID','Loan_Status'])
test_y = test['Loan_Status']
```

[12]:
```
print('shape of training data : ',train_x.shape)
print('shape of testing data : ',test_x.shape)
```

```
shape of training data :  (336, 11)
shape of testing data :  (144, 11)
```

Furthermore, our independent variables (features) still in categorical form so we need to convert them into a dummy/indicator variables (0's &1's)

[13]:
```
# Encode The Data
train_x = pd.get_dummies(train_x)
test_x  = pd.get_dummies(test_x)
print('shape of training data : ',train_x.shape)
print('shape of testing data : ',test_x.shape)
```

```
shape of training data :  (336, 20)
shape of testing data :  (144, 20)
```

2.1.4 Data Scaling:

Now after we converted our independent variables, we need to standardize features by removing the mean and scaling to unit variance using the StandardScalar(). StandardScaler is the industry's go-to algorithm. StandardScaler() standardizes a feature by subtracting the mean and then scaling

to unit variance. Unit variance means dividing all the values by the standard deviation.

```
[15]: # Scale The Data
scaler = StandardScaler().fit(train_x)
train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
```

```
[16]: print(train_x)
```

```
[[ 0.40732648 -0.56153901 -0.73995315 ... -0.61864048  1.19782411
  -0.67419986]
 [-0.33421261 -0.56153901 -0.41848706 ... -0.61864048  1.19782411
  -0.67419986]
 [-0.47203495 -0.56153901 -0.78757776 ... -0.61864048 -0.83484711
   1.4832397 ]
 ...
 [ 0.88375218 -0.56153901  1.16503111 ... -0.61864048 -0.83484711
   1.4832397 ]
 [-0.64236687  0.7542311  -0.31133169 ... -0.61864048 -0.83484711
   1.4832397 ]
 [-0.70624134 -0.56153901 -1.02570079 ... -0.61864048 -0.83484711
   1.4832397 ]]
```

2.2 Machine learning models

2.2.1 Logistic Regression

This is a classification algorithm which uses a logistic function to predict binary outcome (True/False, 0/1, Yes/No) given an independent variable. The aim of this model is to find a relationship between features and probability of particular outcome. The logistic function used is a logit function which is a log of odds in the favor of the event. Logit function develops a s-shaped curve with the probability estimated like a step function.

```
[17]: # Using Logistic Regression Model
LR = LogisticRegression()
LR.fit(train_x,train_y)
predict = LR.predict(test_x)
print('Accuracy Score on test data using Logistic Regression: ', accuracy_score(test_y,predict))
```

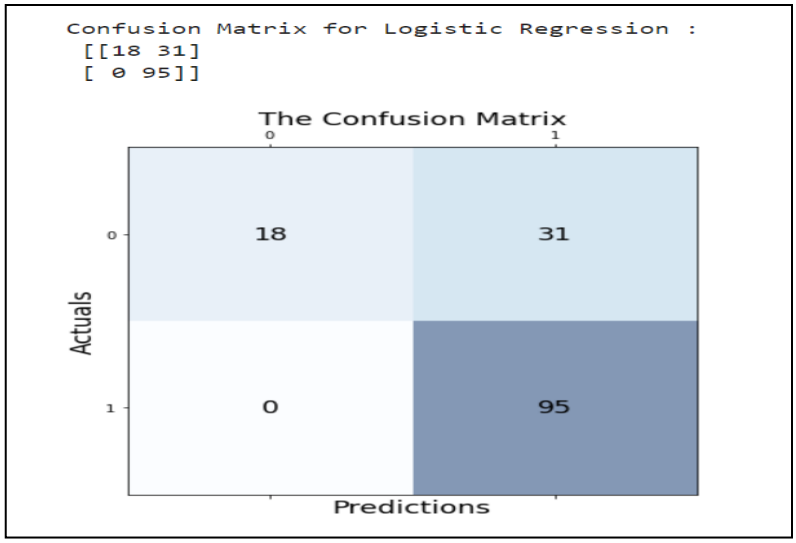
```
Accuracy Score on test data using Logistic Regression:  0.7847222222222222
```

One common way to evaluate the quality of a logistic regression model is to create a confusion matrix, which is a 2×2 table that shows the predicted values from the model vs. the actual values from the test dataset.

```
[18]: # Confusion Matrix for Logistic Regression
cm = metrics.confusion_matrix(test_y, predict)
print('Confusion Matrix for Logistic Regression :\n', cm, '\n')
fig, ax = plt.subplots(figsize=(6.5, 6.5))
ax.matshow(cm, cmap=plt.cm.Blues, alpha=0.5)
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(x=j, y=i,s=cm[i, j], va='center', ha='center', size='xx-large

plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize=18)
plt.title('The Confusion Matrix', fontsize=18)
plt.show()
```

```
Confusion Matrix for Logistic Regression :
 [[18 31]
 [ 0 95]]
```



2.2.2 SVM

Support vector machines are a set of supervised learning methods used for classification, regression, and outliers detection. All of these are common tasks in machine learning.

You can use them to detect cancerous cells based on millions of images or you can use them to predict future driving routes with a well-fitted regression model.

There are specific types of SVMs you can use for particular machine learning problems, like support vector regression (SVR) which is an extension of support vector classification (SVC).

A simple linear SVM classifier works by making a straight line between two classes. That means all of the data points on one side of the line will represent a category and the data points on the other side of the line will be put into a different category. This means there can be an infinite number of lines to choose from.

Now the training set will be fitted to the SVM classifier. To create the SVM classifier, we will import SVC class from Sklearn.svm library. Below is the code for it:

[19]:
```
# Using SVC Model
svc = SVC(kernel="linear")
svc.fit(train_x,train_y)
predict = svc.predict(test_x)
print('Accuracy Score on test data using SVC: ', accuracy_score(test_y,predict))
```

```
Accuracy Score on test data using SVC:  0.7916666666666666
```

In the above code, we have used kernel='linear', as here we are creating SVM for linearly separable data. However, we can change it for non-linear data. And then we fitted the classifier to the training dataset (train_x, train_y)

2.2.3 Decision Trees

This is a supervised machine learning algorithm mostly used for classification problems. All features should be discretized in this model, so that the population can be split into two or more homogeneous sets or subsets. This model uses a different algorithm to split a node into two or more sub-nodes. With the creation of more sub-nodes, homogeneity and purity of the nodes increases with respect to the dependent variable.

[20]:
```
# Using ID3 Model
DT = DecisionTreeClassifier(max_depth=(7), random_state=0)
DT.fit(train_x,train_y)
predict = DT.predict(test_x)
print('Accuracy Score on test data using ID3 : ', accuracy_score(test_y,predict))
```

```
Accuracy Score on test data using ID3 :  0.7638888888888888
```

In the previous image we can see that we used max_depth as our parameter in order to improve our accuracy and prevent any underfitting or overfitting as well as for random_state to control the randomness of the estimator.

2.2.4 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is one of the simplest algorithms used in Machine Learning for regression and classification problem. KNN algorithms use data and classify new data points based on similarity measures (e.g., distance function). Classification is done by a majority vote to its neighbors. The data is assigned to the class which has the nearest neighbors. As you increase the number of nearest neighbors, the value of k, accuracy might increase.

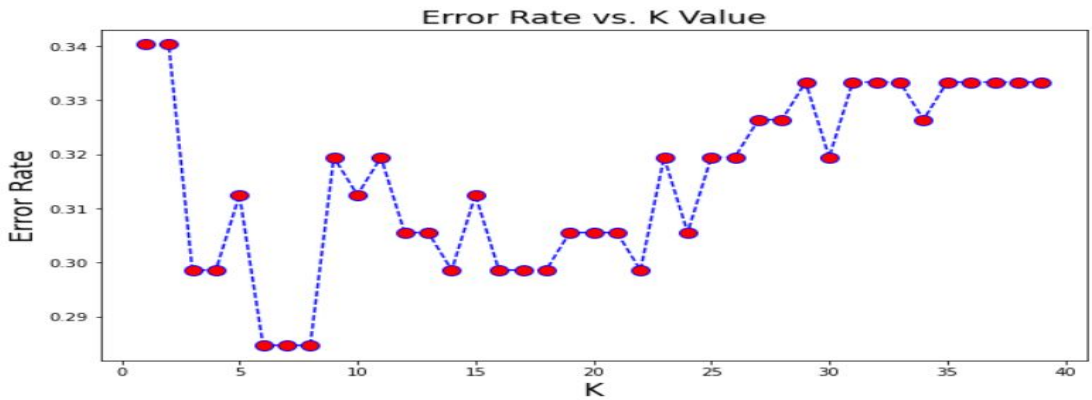
[21]:
```
# Using KNN Model
KNN = KNeighborsClassifier(n_neighbors=7)
KNN.fit(train_x,train_y)
predict = KNN.predict(test_x)
print('Accuracy Score on test data using KNN: ', accuracy_score(test_y,predict))
```

```
Accuracy Score on test data using KNN:  0.7152777777777778
```

To find an optimum value of k we plot a graph of error rate vs k value ranging from 0 to 40.

[22]:
```
# To find a optimum value of K we plot a graph of error rate vs K value ranging from 0 to 40
error_rate = []
for i in range(1,40):
    kNN = KNeighborsClassifier(n_neighbors=i)
    kNN.fit(train_x,train_y)
    predict_i = kNN.predict(test_x)
    error_rate.append(np.mean(predict_i != test_y))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value', fontsize=18)
plt.xlabel('K', fontsize=18)
plt.ylabel('Error Rate', fontsize=18)
plt.show()
```



So, from the graph we can deduce that k=6, k=7 and k=8 has least error rate. Hence, we reapply the algorithm for k=7 for the best accuracy.